Available online at www.sciencedirect.com
ScienceDirect
Discrete Applied Mathematics 155 (2007) 2174–2180
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

Note

Construction of universal one-way hash functions: Tree hashing revisited

Palash Sarkar*

Applied Statistics Unit, Indian Statistical Institute, 203, B.T. Road, Kolkata 700108, India

Received 16 February 2004; received in revised form 14 May 2007; accepted 14 May 2007
Available online 21 May 2007

Abstract

We present a binary tree based parallel algorithm for extending the domain of a universal one-way hash function (UOWHF). For $t \geqslant 2$, our algorithm extends the domain from the set of all n-bit strings to the set of all $((2^t - 1)(n - m) + m)$-bit strings, where m is the length of the message digest. The associated increase in key length is $2m$ bits for $t = 2$; $m(t + 1)$ bits for $3 \leqslant t \leqslant 6$ and $m \times (t + \lfloor \log_2(t - 1) \rfloor)$ bits for $t \geqslant 7$.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Cryptographic hash functions; UOWHF; Binary tree; Parallel computation

1. Introduction

Let $\mathscr{F} = \{h_k\}_{k \in \mathscr{K}}$ be a family of functions where each $h_k : X \to Y$. The set $\mathscr{K}$ is said to be the set of keys and $\mathscr{F}$ is said to be a keyed family of functions. Consider the following adversarial game with respect to $\mathscr{F}$: the adversary outputs an $x \in X$; is then given a randomly chosen key $k \in \mathscr{K}$; and has to output an $x' \in X$ such that $x \neq x'$ and $h_k(x) = h_k(x')$. The family $\mathscr{F}$ is said to be a universal one-way hash function (UOWHF) if it is infeasible for an adversary to win this game. The concept of UOWHF was introduced by Naor and Yung [6] and used by them to show that it is possible to construct secure digital signature schemes based on 1-1, one-way functions.

The notion of collision resistant hash function (CRHF) is more common in cryptography. In this notion, the adversary is first given a k and then has to output x and x' such that $x \neq x'$ and $h_k(x) = h_k(x')$. Note that in the case of UOWHF, the adversary has to commit to the input x even before he knows the function for which a collision is to be found. Intuitively, this makes the adversary's task more difficult and hence UOWHF is considered to be a weaker primitive. In fact, Simon [9] has shown that there is an oracle relative to which UOWHFs exist but not CRHFs.

A systematic study of construction methods for UOWHF was undertaken by Bellare and Rogaway [1]. The approach taken in [1] is to start with a UOWHF $\{h_k\}_{k \in \mathscr{K}}$, where the domain X_1 of h_k consists of short fixed length strings and then obtain methods to construct another UOWHF $\{H_p\}_{p \in \mathscr{P}}$ where the domain X_2 of H_p consists of long fixed length strings. Such a method is called a domain extending algorithm. Further, it is shown in [1], that extending the domain also requires an increase in the length of the keys.

* Tel.: +91 33 2575 2830; fax: +91 33 2577 3104.
E-mail address: palash@isical.ac.in.

0166-218X/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2007.05.005

The most important method in [1] is a tree based construction. For binary trees with $t \geqslant 2$ levels, the BR construction extends the domain from n-bit strings to $((2^t - 1)(n - m) + m)$-bit strings and requires an associated increase in key length by $2m(t - 1)$ bits, where the range Y of the hash functions consists of m-bit strings. Later, Shoup [8] provided a sequential construction which requires an increase in key length of mt bits for extending the domain by the same amount. Thus, the sequential algorithm has a smaller key length expansion compared to the earlier binary tree based algorithm. Mironov [5] provided an alternative proof of correctness of Shoup's result and showed that the key length expansion is the best possible for any sequential algorithm.

The essential computational task in a domain extending algorithm consists of executing h_k several times. In a binary tree based algorithm, the different invocations of h_k are organised in a binary tree. This opens up the possibility of parallel execution using more than one processors. Such a parallel algorithm will have several parallel rounds. In each round, all (or some) of the processors would operate in parallel and produce outputs to be used in the next round. Since more than one processors work simultaneously, the number of parallel rounds will be less than the total number of invocations of h_k made by the algorithm. On the other hand, a sequential algorithm using one processor will require time proportional to the total number of invocations of h_k. There are standard ways to convert a binary tree structure into an efficient parallel algorithm (see [3]). Thus, even though the binary tree algorithm of [1] has a larger key length expansion compared to the sequential algorithm of [8], it will be more efficient to implement when more than one processors are available.

In this work, we show that a modification of the binary tree algorithm of [1] results in a smaller key length expansion while retaining the advantage of parallelism. The key length expansion of our algorithm is $2m$ bits for $t = 2$; $m(t + 1)$ bits for $3 \leqslant t \leqslant 6$ and $m \times (t + \lfloor \log_2(t - 1) \rfloor)$ bits for $t \geqslant 7$. This is an improvement over the algorithm in [1] though it is more than the key length expansion achieved in [8]. The improvement in key length expansion over [1] is achieved by combining the construction in [1] with the construction in [8]. The proof of correctness of our algorithm employs the technique used in [5] rather than the technique in [8].

2. Preliminaries

Let $\{h_k\}_{k \in \mathcal{K}}$ be a keyed family of hash functions, where each $h_k : X \to Y$. *In this paper we require* $n \geqslant 2m$. Consider the following adversarial game.

1. Adversary chooses an $x \in X$.
2. Adversary is given a k which is chosen uniformly at random from $\mathcal{K}$.
3. Adversary has to find $x' \in X$ such that $x \neq x'$ and $h_k(x) = h_k(x')$.

We say that $\{h_k\}_{k \in \mathcal{K}}$ is a universal one way hash family (UOWHF) if the adversary has a negligible probability of success with respect to any randomized polynomial time strategy. A strategy $\mathcal{A}$ for the adversary runs in two stages. In the first stage $\mathcal{A}^{\text{guess}}$, the adversary finds the x to which he has to commit in Step 1. It also produces some auxiliary state information s. In the second stage $\mathcal{A}^{\text{find}}(x, k, s)$, the adversary either finds a x' which provides a collision for h_k or it reports failure. Both $\mathcal{A}^{\text{guess}}$ and $\mathcal{A}^{\text{find}}(x, k, s)$ are randomized algorithms. The success probability of the strategy is measured over the random choices made by $\mathcal{A}^{\text{guess}}$ and $\mathcal{A}^{\text{find}}(x, k, s)$ and the random choice of k in step 2 of the game. We say that $\mathcal{A}$ is an (ε, a)-strategy if the success probability of $\mathcal{A}$ is at least ε and it invokes the hash function h_k at most a times. In this case we say that the adversary has an (ε, a)-strategy for $\{h_k\}_{k \in \mathcal{K}}$. Note that we do not include time as an explicit parameter though it would be easy to do so.

In this paper we are interested in extending the domain of a UOWHF. Thus, given a UOWHF $\{h_k\}_{k \in \mathcal{K}}$, with $h_k : \{0, 1\}^n \to \{0, 1\}^m$ and a positive integer $L \geqslant n$, we would like to construct another UOWHF $\{H_p\}_{p \in \mathcal{P}}$, with $H_p : \{0, 1\}^L \to \{0, 1\}^m$. We say that the adversary has an (ε, a)-strategy for $\{H_p\}_{p \in \mathcal{P}}$ if there is a strategy $\mathcal{B}$ for the adversary with probability of success at least ε and which invokes the hash function h_k at most a times. Note that H_p is built using h_k and hence while studying strategies for H_p we are interested in the number of invocations of the hash function h_k.

The correctness of our construction will essentially be a Turing reduction. (See [10] for more on this approach.) We will show that if there is an (ε, a)-strategy for $\{H_p\}_{p \in \mathcal{P}}$, then there is an (ε_1, a_1)-strategy for $\{h_k\}_{k \in \mathcal{K}}$, where a_1 is not much larger than a and ε_1 is not significantly smaller than ε. This will show that if $\{h_k\}_{k \in \mathcal{K}}$ is a UOWHF, then so is $\{H_p\}_{p \in \mathcal{P}}$.

The key length for the base hash family $\{h_k\}_{k \in \mathcal{K}}$ is $\lceil \log_2 |\mathcal{K}| \rceil$. On the other hand, the key length for the family $\{H_p\}_{p \in \mathcal{P}}$ is $\lceil \log_2 |\mathcal{P}| \rceil$. Thus, increasing the size of the input from n bits to L bits results in an increase of the key size

by an amount $\lceil \log_2 |\mathscr{P}| \rceil - \lceil \log_2 |\mathscr{K}| \rceil$. From a practical point of view a major motivation is to minimise this increase in the key length.

3. Known constructions

We briefly discuss the sequential construction by Shoup [8] and the tree construction by Bellare–Rogaway [1].

3.1. Sequential construction

The Merkle–DamgAArd construction is a well-known method for extending the domain of a collision resistant hash functions. However, Bellare and Rogaway [1] showed that the construction does not directly work in the case of UOWHF. In [8], Shoup presented a modification of the MD construction. We briefly describe the Shoup construction as presented in [5].

Let $\{h_k\}_{k\in\mathscr{K}}$, with $h_k : \{0,1\}^n \to \{0,1\}^m$ and $\mathscr{K} = \{0,1\}^K$ be the UOWHF whose domain is to be extended. Let x be the input to H_p with $|x| = r(n-m)$. We define $p = k\|m_0\|m_1\|\ldots\|m_{l-1}$ where $l = 1 + \lfloor \log r \rfloor$ and m_i are m-bit randomly chosen binary strings called masks. The increase in key length is lm bits. The output of H_p is computed by the following algorithm. Define $v(i) = j$ if $2^j | i$ and $2^{j+1} \nmid i$.

1. Let $x = x_1\|x_2\|\ldots\|x_r$, where $|x_i| = n - m$.
2. Let IV be an n-bit initialisation vector.
3. Define $z_0 = \mathrm{IV}$, $s_0 = z_0 \oplus m_0$.
4. For $1 \leqslant i \leqslant r$, define $z_i = h_k(s_{i-1}\|x_i)$ and $s_i = z_i \oplus m_{v(i)}$.
5. Define z_r to be the output of $H_p(x)$.

A proof of correctness of the construction was described by Shoup in [8]. In a later work, Mironov [5] provided an alternative correctness proof. More importantly, in [5] it was shown that the amount of key length expansion is the minimum possible for the construction to be correct.

3.2. Tree based construction

In [1], Bellare and Rogaway described a tree based construction for extending UOWHF. We briefly describe the construction for binary trees. As before, let $\{h_k\}_{k\in\mathscr{K}}$ be the UOWHF whose domain is to be extended.

There are $2^t - 1$ processors $P_1, \ldots, P_{2^t-1}$ connected in a full binary tree of t levels numbered $1, \ldots, t$ and $2^t - 1$ nodes. The processors $P_{2^{i-1}}, \ldots, P_{2^i-1}$ are at level i. The arcs in the binary tree point towards the parent, i.e., the arcs are of the form $(2i, i)$ and $(2i+1, i)$. Each processor is capable of computing the function h_k for any $k \in \mathscr{K}$, i.e., $P_i(k, x) = h_k(x)$, for an n-bit string x. *In the rest of the paper we will always assume that* $t \geqslant 2$. By *level*(i) we denote the level of the tree to which P_i belongs. Thus, *level*$(i) = j$ if $2^{j-1} \leqslant i \leqslant 2^j - 1$. It is clear that all the nodes at the same level can work in parallel.

Denote the extended domain UOWHF to be $\{H_p\}_{p\in\mathscr{P}}$. The input to the function H_p is x of length $2^{t-1}n + (2^{t-1} - 1)(n - 2m)$. The key p for the function H_p is formed out of the key k for the function h_k plus some additional m-bit strings called *masks*. In the Bellare–Rogaway (BR) algorithm, we have $p = k\|\alpha_1\|\beta_1\|\ldots\|\alpha_{t-1}\|\beta_{t-1}$, where α_i's and β_j's are randomly chosen m-bit strings. The computation of the function $H_p(x)$ is done in the following manner.

BR Construction:

1. Write $x = x_1\|x_2\|\ldots\|x_{2^t-1}$, where $|x_1| = \cdots = |x_{2^{t-1}-1}| = n - 2m$ and $|x_{2^{t-1}}| = \cdots = |x_{2^t-1}| = n$; (note $|x| = 2^t(n-m) - (n-2m)$.)
2. for $2^{t-1} \leqslant i \leqslant 2^t - 1$, do in parallel
   (a) compute $z_i = P_i(k, x_i) = h_k(x_i)$;
   (b) set $s_i = z_i \oplus \alpha_{t-1}$ if i is even, and set $s_i = z_i \oplus \beta_{t-1}$ if i is odd;
3. for $j = t - 1$ down to 2 do
   - for $i = 2^{j-1}$ to $2^j - 1$ do in parallel
   (a) $z_i = P_i(k, s_{2i}\|s_{2i+1}\|x_i) = h_k(s_{2i}\|s_{2i+1}\|x_i)$.
   (b) set $s_i = z_i \oplus \alpha_{j-1}$ if i is even and set $s_i = z_i \oplus \beta_{j-1}$ if i is odd;

4. define the output of $H_p(x)$ to be $h_k(s_2\|s_3\|x_1)$.

We note that in the original algorithm in [1], the strings $x_1, \ldots, x_{2^{t-1}-1}$ were defined to be empty strings. The amount of key length expansion is $2(t-1)m$ bits for a tree with t levels. Thus, $2(t-1)$ masks each of length m bits are required by the construction.

Remark. The processors $P_1, \ldots, P_{2^t-1}$ are mentioned for the sake of clarity. In practice, the binary tree algorithm described above can be carried out by $\kappa \leqslant 2^t/t$ processors in not more than $t + \lceil 2^t/\kappa \rceil$ rounds. This is a basic fact in parallel computation (see, for example, [3]).

4. Improved tree based construction

As in the BR algorithm, assume that $\{H_p\}_{p\in\mathscr{P}}$ is to be constructed from $\{h_k\}_{k\in\mathscr{K}}$ using the $2^t - 1$ processors $P_1, \ldots, P_{2^t-1}$. As before, the input to the function H_p is a string x of length $2^{t-1}n + (2^{t-1} - 1)(n - 2m)$ and the key p for the function H_p is formed out of the key k for the function h_k plus some masks. The definition of these masks in our algorithm is different from that in the BR algorithm.

For convenience in describing the algorithm we divide these masks into two *disjoint* sets $\Gamma = \{\alpha_1, \ldots, \alpha_{t-1}\}$ and $\Delta = \{\beta_0, \ldots, \beta_{l-1}\}$, where $l = 1 + \lfloor \log_2(t-1) \rfloor$. Recall that for integer i, the function $v(i) = j$ if $2^j | i$ and $2^{j+1} \nmid i$. The new algorithm can be described by simply changing Lines 2(b) and 3(b) in the following manner.

Improved Tree Construction (*ITC*):

2(b) set $s_i = z_i \oplus \beta_0$ if i is even and set $s_i = z_i \oplus \alpha_1$ if i is odd;
3(b) set $s_i = z_i \oplus \beta_{v(t-j+1)}$ if i is even and set $s_i = z_i \oplus \alpha_{t-j+1}$ if i is odd.

We provide an explanation of the construction. Let $P = P_{i_r} P_{i_{r-1}} \ldots P_{i_1}$ be a path of processors of length r from the leaf node P_{i_r} to some internal node P_{i_1} which is obtained by following only left links, i.e., $level(i_r) = t$ and $i_{j+1} = 2i_j$ for $j = 1, \ldots, r-1$. The arcs (i_{j+1}, i_j) in the path are assigned masks according to the Shoup construction. Let S be the set of arcs $\{(2i_1 + 1, i_1), (2i_2 + 1, i_2), \ldots, (2i_{r-1} + 1, i_{r-1})\}$. The construction also ensures that no two arcs in S get the same mask.

Proposition 1. *The following are true for algorithm ITC.*

1. *t parallel rounds are required to compute the output.*
2. *The function h_k is invoked $2^t - 1$ times.*
3. *The amount of key length expansion $(|p| - |k|)$ is $m(t + \lfloor \log_2(t-1) \rfloor)$ bits.*

Proof. (1) Step 2 of ITC is one parallel round. Step 3 requires $(t-2)$ parallel rounds and Step 4 requires one round. Hence, a total of t rounds are required.

(2) There are $2^t - 1$ processors and each processor invokes the function h_k exactly once. Hence, h_k is invoked exactly $2^t - 1$ times.

(3) The amount of key length expansion is $m \times |\Gamma \cup \Delta|$. By definition $|\Gamma| = t - 1$ and $|\Delta| = 1 + \lfloor \log_2(t-1) \rfloor$. Also $\Gamma \cap \Delta = \emptyset$. □

Remark. The amount of expansion in the BR construction is $2(t-1)m$ bits. Thus, with respect to key length expansion ITC is an improvement over the BR construction.

Theorem 2 (*Security reduction for H_p*). *If there is an (ε, a) winning strategy $\mathscr{A}$ for $\{H_p\}_{p\in\mathscr{P}}$, then there is an $(\varepsilon/(2^t - 1), a + 2(2^t - 1))$ winning strategy $\mathscr{B}$ for $\{h_k\}_{k\in\mathscr{K}}$. Consequently, $\{H_p\}_{p\in\mathscr{P}}$ is a UOWHF if $\{h_k\}_{k\in\mathscr{K}}$ is a UOWHF.*

Proof. We describe the two stages of the strategy $\mathscr{B}$ as follows:

$\mathscr{B}^{\text{guess}}$: (output (y, s), with $|y| = n$.)

1. Run $\mathscr{A}^{\text{guess}}$ to obtain $x \in \{0,1\}^L$ and state information s'.
2. Choose an i uniformly at random from the set $\{1, \ldots, 2^t - 1\}$.

3. Write $x = x_1 \| \ldots \| x_{2^t-1}$, where $|x_1| = \cdots = |x_{2^{t-1}-1}| = n - 2m$ and $|x_{2^{t-1}}| = \cdots = |x_{2^t-1}| = n$.
4. If $2^{t-1} \leqslant i \leqslant 2^t - 1$, set $y = x_i$; u_1, u_2 to be the empty string and $s = (s', i, u_1, u_2)$. Output (y, s) and stop.
5. If $1 \leqslant i \leqslant 2^{t-1} - 1$, then choose two strings u_1 and u_2 uniformly at random from the set $\{0, 1\}^m$. Set $y = u_1 \| u_2 \| x_i$ and $s = (s', i, u_1, u_2)$. Output (y, s) and stop.

At this point the adversary is given a k which is chosen uniformly at random from the set $\mathscr{K} = \{0, 1\}^K$. The adversary then runs $\mathscr{B}^{\text{find}}$ which is described below.

$\mathscr{B}^{\text{find}}(y, k, s)$: (Note $s = (s', i, u_1, u_2)$.)

1. Define the masks $\alpha_1, \ldots, \alpha_{t-1}, \beta_0, \ldots, \beta_{l-1}$ by executing algorithm $\mathsf{MDef}(i, u_1, u_2)$ (called the mask defining algorithm; will be described later). This defines the key p for the function H_p. Here $p{=}k \| \alpha_1 \| \ldots \| \alpha_{t-1} \| \beta_0 \| \ldots \| \beta_{l-1}$, where $l = \lfloor \log_2(t-1) \rfloor + 1$.
2. Run $\mathscr{A}^{\text{find}}(x, p, s')$ to obtain x'.
3. Let v and v' be the inputs to processor P_i corresponding to the strings x and x', respectively. Denote the corresponding outputs by z_i and z_i'. If $z_i = z_i'$ and $v \neq v'$, then output v and v', else output "failure".

Note that Step 3 either detects a collision or reports failure. We now lower bound the probability of success. But first we have to specify the mask defining algorithm.

The task of the mask defining algorithm MDef is to define the masks $\alpha_1, \ldots, \alpha_{t-1}, \beta_0, \ldots, \beta_{l-1}$ (and hence p) so that the input to processor P_i is y. Note that the masks are not defined until the key k is given to the adversary. Once the key k is specified we extend it to p such that the extension is "consistent" with the input y to P_i to which the adversary has already committed. Another point that one has to be careful about is to ensure that the key p is chosen uniformly at random from the set $\mathscr{P}$, i.e., the masks α_i and β_j are chosen independently and uniformly to be m-bit strings.

The mask defining algorithm MDef is given below. The algorithm uses an array $A[..]$ of length at most $(t-1)$ whose entries are pairs of the form (j, v) where j is an integer in the range $1 \leqslant j \leqslant 2^t - 1$ and v is an m-bit string.

Algorithm $\mathsf{MDef}(i, u_1, u_2)$

(Note: i was chosen by $\mathscr{B}^{\text{guess}}$ in Step 2. The strings u_1 and u_2 were chosen by $\mathscr{B}^{\text{guess}}$ either in Step 4 or in Step 5.)

1. If $2^{t-1} \leqslant i \leqslant 2^t - 1$, then randomly define the masks $\alpha_1, \ldots, \alpha_{t-1}, \beta_0, \ldots, \beta_{l-1}$ and exit.
2. Append $(2i + 1, u_2)$ to the array A.
3. Let $j = t - level(i)$, $j_1 = j - 2^{v(j)}$ and $i_1 = 2^{j-j_1} i$.
4. Randomly define all undefined masks in the set $\beta_{v(j_1+1)}, \ldots, \beta_{v(j-1)}$.
5. If $j_1 = 0$, then $z_{i_1} = h_k(x_{i_1})$,
6. else
   (a) randomly choose u, v in $\{0, 1\}^m$.
   (b) Append $(2i_1 + 1, v)$ to the array A.
   (c) $z_{i_1} = h_k(u \| v \| x_{i_1})$.
7. For $j_2 = j_1 + 1, \ldots, j - 1$ do
   (a) $i_2 = 2^{j-j_2} i$.
   (b) $s_{2i_2} = z_{2i_2} \oplus \beta_{v(j_2)}$.
   (c) Randomly choose w in $\{0, 1\}^m$.
   (d) Append $(2i_2 + 1, w)$ to the array A.
   (e) $z_{i_2} = h_k(s_{2i_2} \| w \| x_{i_2})$.
8. $\beta_{v(j)} = z_{2i} \oplus u_1$.
9. If $j_1 > 0$, then $u_1 = u$, $u_2 = v$, $j = j_1$ and go to Step 2.
10. Randomly define all as yet undefined masks β_i, $0 \leqslant i \leqslant l - 1$.
11. Sort the array A in descending order based on the first component of each entry (j, v).
12. For $i_1 = 1$ to $t - level(i)$ do
    (a) Let $(l, u) = A[i_1]$.
    (b) Compute z_l to be the output of processor P_l. (This can be done, since at this point all masks used in the subtree rooted at l have already been defined.)
    (c) Let $j = t - level(l) + 1$.
    (d) Define $\alpha_j = z_l \oplus u$.
13. Randomly define all as yet undefined masks α_j, $1 \leqslant j \leqslant t - 1$.

Intuitively, algorithm MDef applies the mask reconstruction algorithm for the Shoup construction along the path $P_{\imath_r}, P_{\imath_{r-1}}, \ldots, P_{\imath_1}$, where $\imath_1 = i, \imath_j = 2^{j-1}i$ and $level(\imath_r) = t$. This defines the masks $\beta_{v(t-level(\imath_j))}$ for $1 \leqslant j < r$. To do this the algorithm guesses the inputs that the processors $P_{\imath_1}, \ldots, P_{\imath_{r-1}}$ obtain from their right descendants. These inputs along with the proper processor numbers are added to the array A. Once the definition of the β masks are complete, the algorithm begins the task of defining the α masks. The first element of the array A is $(2\imath_{r-1}+1, u)$ for some m-bit string u and we are required to define α_1. The processor $P_{2\imath_{r-1}+1}$ is at the leaf level and applies h_k to $x_{2\imath_{r-1}+1}$ to produce $z_{2\imath_{r-1}+1}$. Now α_1 is defined to be the XOR of u and $z_{2\imath_{r-1}+1}$. Suppose for some $2 \leqslant j \leqslant r$, the masks $\alpha_1, \ldots, \alpha_{j-1}$ has already been defined. The current element of the array A is $(2\imath_{r-j}+1, u)$ for some m-bit string u. At this point all masks present in the subtree rooted at processor $P_{2\imath_{r-j}+1}$ have already been defined. Thus, the input to processor $P_{2\imath_{r-j}+1}$ is known. Hence, processor $P_{2\imath_{r-j}+1}$ applies the hash function h_k to its input to obtain the string $z_{2\imath_{r-j}+1}$. The mask α_j is now defined to be the XOR of u and $z_{2\imath_{r-j}+1}$.

Notice that this procedure ensures that the input to processor P_i is the string y to which $\mathscr{B}^{\text{guess}}$ has committed. We now argue that the masks are chosen randomly from the set $\{0,1\}^m$. For this we note that in MDef each mask is either chosen to be a random string or is obtained by XOR with a random string. Hence, all the masks are random strings from the set $\{0,1\}^m$. Also k is a random string and hence p is a randomly chosen key from the set $\mathscr{P}$.

Suppose x and x' collide for the function H_p. Then there must be a j in the range $1 \leqslant j \leqslant 2^t - 1$ such that processor P_j provides a collision for the function h_k. (Otherwise it is possible to prove by a backward induction that $x = x'$.) The probability that $j = i$ is $1/(2^t - 1)$. Hence, if the success probability of $\mathscr{A}$ is at least ε, then the success probability of $\mathscr{B}$ is at least $\varepsilon/(2^t - 1)$. Also the number of invocations of h_k by $\mathscr{B}$ is equal to the number of invocations of h_k by $\mathscr{A}$ plus at most $2(2^t - 1)$. This completes the proof. □

4.1. Improvement on Algorithm ITC for $t = 5, 6$

Algorithm ITC uses two disjoint sets of masks Γ and Δ. For $t = 5, 6$, we have $\Gamma = \{\alpha_1, \ldots, \alpha_{t-1}\}$ and $\Delta = \{\beta_0, \beta_1, \beta_2\}$. This results in a total of $t + 2$ distinct masks. The next result shows that $t + 1$ masks are sufficient for these values of t.

Theorem 3. *For $t = 5, 6$, it is possible to properly extend a UOWHF $\{h_k\}_{k \in \mathscr{K}}$ to a UOWHF $\{H_p\}_{p \in \mathscr{P}}$ using a processor tree of $2^t - 1$ processors and requiring exactly $t + 1$ masks.*

Proof. The algorithm is same as Algorithm ITC with the following small modification. In Algorithm ITC the sets $\Gamma = \{\alpha_1, \ldots, \alpha_{t-1}\}$ and $\Delta = \{\beta_0, \beta_1, \beta_2\}$ are disjoint. We remove this disjointness by setting $\alpha_1 = \beta_2$. This results in a total of $t + 1$ masks.

We have to show that setting $\alpha_1 = \beta_2$ does not affect the correctness of the construction. More precisely, we have to provide a security reduction similar to that of Theorem 2. A close examination of the proof of Theorem 2 shows that the only part of the proof which will be affected by setting $\alpha_1 = \beta_2$ is the mask defining algorithm. Thus, it is sufficient to describe a proper mask defining algorithm. We describe the mask defining algorithm for $t = 6$ which will also cover the case $t = 5$.

Let the processors for $t = 6$ be $P_1, \ldots, P_{63}$. Suppose the output of $\mathscr{B}^{\text{guess}}$ is $(y, s = (s', i, u_1, v_1))$. If $i \geqslant 4$, then the mask defining algorithm of Theorem 2 is sufficient to define all the masks. This is because of the fact that in Algorithm ITC the mask β_2 does not occur in the subtree rooted at i and hence we are required to define only α_1. The problem arises when we have to define both α_1 and β_2 using Algorithm MDef. Since in this case $\alpha_1 = \beta_2$, defining one will define the other. Thus, we have to ensure that this particular mask is not redefined.

There are three values of i that we have to consider, namely $i = 1, 2$ and 3. The case $i = 1$ is the most general as it requires us to consider the full tree of 63 processors. The other two cases, $i = 2$ and $i = 3$ are simpler and are essentially the same. These two cases require defining the masks for a tree with 31 processors which correspond to $t = 5$. Here we only describe the case of $i = 1$.

1. Randomly choose two m-bit strings u_2 and v_2. Define $\beta_0 = u_1 \oplus h_k(u_2 \| v_2 \| x_2)$.
2. Randomly choose two m-bit strings u_3 and v_3.
   (a) Set $w_1 = h_k(u_3 \| v_3 \| x_8)$.
   (b) Set $w_2 = w_1 \oplus \beta_0$.
   (c) Randomly choose an m-bit string v_4.

(d) Set $w_3 = h_k(w_2 \| v_4 \| x_4)$.
(e) Define $\beta_2 = u_2 \oplus w_3$.

3. (a) Set $w_4 = \beta_0 \oplus h_k(x_{32})$.
   (b) Set $w_5 = \alpha_1 \oplus h_k(x_{33})$. (Note that $\alpha_1 = \beta_2$ has been defined in Step 2(e).)
   (c) Define $\beta_1 = u_3 \oplus h_k(w_4 \| w_5 \| x_{16})$.
4. Compute the output of processor P_{17} and call it w_6. Define $\alpha_2 = w_6 \oplus v_3$.
5. Compute the output of processor P_9 and call it w_7. Define $\alpha_3 = w_7 \oplus v_4$.
6. Compute the output of processor P_5 and call it w_8. Define $\alpha_4 = w_8 \oplus v_2$.
7. Compute the output of processor P_3 and call it w_9. Define $\alpha_5 = w_9 \oplus v_2$.

It is not difficult to verify that the above algorithm properly defines all the masks. Further, each mask is obtained by XOR with a random m-bit string and hence the concatenation of the $(t + 1)$ different masks is a random bit string of length $m(t + 1)$. This completes the proof of the theorem. □

5. Conclusion

In this paper, we have considered the problem of extending the domain of a UOWHF using a binary tree algorithm. As shown in [1] this requires an expansion in the length of the key to the hash function. To extend the domain from n-bit strings to $((2^t - 1)(n - m) + m)$-bit strings, our algorithm makes a key length expansion of $2m$ bits for $t = 2$; $m(t + 1)$ bits for $3 \leqslant t \leqslant 6$ and $m(t + \lfloor \log_2(t - 1) \rfloor)$ for $t \geqslant 7$ where m is the length of the message digest. The binary tree algorithm of Bellare and Rogaway [1] requires a key length expansion of $2m(t - 1)$ with the same parameters. Hence, the key length expansion in our algorithm is smaller. However, it is greater than the sequential algorithm due to Shoup [8], which requires a key length expansion of mt for the same parameters. On the other hand, the advantage of the BR and our binary tree algorithm is that it is parallelizable while Shoup's algorithm is not.

Acknowledgement

We would like to thank the reviewers for carefully reading an earlier version of the paper and providing detailed comments which helped in improving the presentation of the paper.

References

[1] M. Bellare, P. Rogaway, Collision-resistant hashing: towards making UOWHFs practical, in: Proceedings of Crypto 1997, Lecture Notes in Computer Science, vol. 1294, Springer, Berlin, 1997, pp. 470–484.
[3] J. JáJá, An Introduction to Parallel Algorithms, Addison-Wesley, Reading, MA, 1992.
[5] I. Mironov, Hash functions: from Merkle–DamgAArd to Shoup, in: Proceedings of Eurocrypt 2001, Lecture Notes in Computer Science, vol. 2045, Springer, Berlin, 2001, pp. 166–181.
[6] M. Naor, M. Yung, Universal one-way hash functions and their cryptographic applications, in: Proceedings of the 21st Annual Symposium on Theory of Computing, ACM, New York, 1989, pp. 33–43.
[8] V. Shoup, A composition theorem for universal one-way hash functions, in: Proceedings of Eurocrypt 2000, Lecture Notes in Computer Science, vol. 1807, Springer, Berlin, 2000, pp. 445–452.
[9] D. Simon, Finding collisions on a one-way street: can secure hash function be based on general assumptions?, in: Proceedings of Eurocrypt 1998, Lecture Notes in Computer Science, vol. 1403, Springer, Berlin, 1998, pp. 334–345.
[10] D.R. Stinson, Some observations on the theory of cryptographic hash functions, Des. Codes Cryptogr. 38 (2) (2006) 259–277.